UNITED STATES DEPARTMENT OF COMMERCE
Patent and Trademark Office
Address: COMMISSIONER OF PATENTS AND TRADEMARKS
Washington, D.C. 20231

APPLICATION NO.	FILING DATE	FIRST NAMED INVENTOR	ATTORNEY DOCKET NO.
08/738,659	10/30/96	MOTOYAMA	T 5244-051-2X-

LM51/0816

OBLON SPIVAK MCCLELLAND MAIER AND
NEUSTADT
FOURTH FLOOR
1755 JEFFERSON DAVIS HIGHWAY
ARLINGTON VA 22202

EXAMINER	
LUU, L	
ART UNIT	PAPER NUMBER
2756	

DATE MAILED: 08/16/99

Please find below and/or attached an Office communication concerning this application or proceeding.

Commissioner of Patents and Trademarks

UNITED STATES DEPARTMENT OF COMMERCE
Patent and Trademark Office
ASSISTANT SECRETARY AND COMMISSIONER OF
PATENTS AND TRADEMARKS
Washington, D.C. 20231

**BEFORE THE BOARD OF PATENT APPEALS
AND INTERFERENCES**

RECEIVED
AUG 16 1999
Group 2700

Paper No. 20

Application Number: 08/738,659

Filing Date: 10/30/96

Appellant(s): Motoyama

[illegible]

James J. Kulbaski

For Appellant

SUPPLEMENTAL EXAMINER ANSWER IN RESPONSE TO REPLY BRIEF

The reply brief filed on 06/07/99 has been filed and considered.

This is in response to appellant's reply brief on appeal filed on 06/07/99, a Supplemental Examiner's Answer is set forth below:

(A) Applicant argues that Kraslavsky et al (Kraslavsky) is directed to a high speed system for transmitting real-time or near-real-time status information. Modifying Kraslavsky to operate using Internet electronic mail which may be quite slow would be contrary to the teachings of Kraslavsky.

As to point (A), Kraslavsky's invention is to eliminate the necessity of dedicating a personal computer to manage a peripheral such as a printer by providing an apparatus for interfacing the printer to a LAN (Kraslavsky, Field of the Invention, col. 1 lines 14-23). Obviously, Kraslavsky does not teach improving system speed by connecting the printer on LAN because a computer has a printer connected directly on the computer's parallel port provides the best real-time or near real time for status information compare to printer connected on LAN. Therefore, combining Cohn's teaching to use Internet electronic mail to manage Kraslavsky's peripheral without dedicating a personal computer enhances Kraslavsky's invention.

(B) Applicant argues that electronic mail has historically been for the transmission of text messages among computer users. Therefore, one of ordinary skill in the art at the time the invention was made would not utilize e-mail to transmit the type of information which is exchanged in Kraslavsky.

As to point (B), Cohn teaches various source and destination message systems that comprise voice mail, electronic mail, facsimile transmission, or video transmission capabilities that can communicate compound message to each others using Internet electronic mail message format (col. 8 lines 36-65, and col. 15 lines 65 - col. 16 line 36). Therefore, it would have been obvious to one of ordinary skill in the art at the time of the invention was made to utilize Internet electronic mail to transmit many different types of information including the type of information which is exchanged in Kraslavsky among various source and destination message systems because it would allow any type of messages to be transferred globally between any devices.

(C) Applicant argues that Kraslavsky already provide a well known standard such as TCP/IP (e.g., a global standard) for exchanging messages and performing electronic communication. Thus, as Kraslavsky already contain TCP/IP global standard for exchanging messages, it would not have been obvious to modify Kraslavsky to utilize Internet electronic mail to transfer messages globally between devices as this function is already available within Kraslavsky.

As to point (C), Applicant incorrectly thinks that Internet electronic mail is not part of

TCP/IP standard. It is well known by one of ordinary skill in the art at the time the invention was made that Internet electronic mail is TCP/IP standard for electronic mail service and this well known feature is clearly documented by Doughlas E. Commer's book titled "Internetworking With TCP/IP". Examiner enclosed a copy of chapter 25 of the book that specifically teach Electronic Mail in TCP/IP for reference (Appendix A, Commer, section 25.6 TCP/IP Standards For Electronic Mail Service begins on page 438, and section 25.9 Simple Mail Transfer Protocol (SMTP) begins on page 440).

(D) Applicant argues that a reference cannot be modified to destroy its purpose.

As to point (D), Examiner has discussed in items (A)-(C) above; it is obvious to one of ordinary skill in the art at the time the invention was made that the combining of Cohn's teaching into Kraslavsky's invention enhance Kraslavsky's invention as discussed above.

(E) Applicant argues that electronic mail messages are not connection mode messages, the system would not use a connection mode of communication. Thus, it is completely improper to find that the combination of Kraslavsky and Cohn provide both a teaching in which an Internet electronic mail message is transmitted, and also under specific circumstances which are recited in the claims, a connection mode message is transmitted.

As to point (E), It is well known that the Internet provides both connection and connectionless mode of communication; however, Internet electronic mail message is supported by SMTP protocol using TCP connection protocol which is connection mode of communication as described in SMTP Simple Mail Transfer Protocol by Stevens' reference which was submitted by applicant (Appendix B, Sevens, SMTP: Simple Mail Transfer Protocol pages 441-444). Connectionless mode of communication is well known that it does not provide return messages or acknowledgments. In the contrary, Internet electronic mail supported by SMTP protocol using TCP connection protocol provides acknowledgments and it clearly described by Stevens. Therefore, Kraslavsky and Cohn teach transmitting Internet electronic mail message, and also teach transmitting a connection mode message under specific circumstances.

For the reasons above, it is believed that the rejections should be sustained.

Respectfully submitted,

[illegible]

Le Hien Luu

August 09, 1999

Appendix A

SN 08/738,659

“Internetworking With TCP/IP” Vol 1: Principles, Protocols, and Architecture”
Third Edition
Douglas E. Comer, Prentice Hall, pp. 433-446.

Internetworking With TCP/IP
Vol I:
Principles, Protocols, and Architecture
Third Edition

DOUGLAS E. COMER

Department of Computer Sciences
Purdue University
West Lafayette, IN 47907

PRENTICE HALL
Upper Saddle River, New Jersey 07458

Library of Congress Cataloging-in-Publication Data

Comer, Douglas
Internetworking with TCP/IP / Douglas E. Comer. -- 3rd ed.
p. cm.
Includes bibliographical references and index.
Contents: v. 1. Principles, protocols, and architecture
ISBN 0-13-216987-8 (v. 1)
1. TCP/IP (Computer network protocol) 2. Client/server computing.
3. Internetworking (Telecommunication) I. Title.
TK5105.585.C66 1995
005.2--dc20
95-1830
CIP

Acquisitions editor: ALAN APT
Production editor: IRWIN ZUCKER
Cover designer: WENDY ALLING JUDY
Buyer: LORI BULWIN
Editorial assistant: SHIRLEY MCGUIRE

Printed in the United States of America

10 9 8 7

ISBN 0-13-216987-8

Prentice-Hall International (UK) Limited, London
Prentice-Hall of Australia Pty. Limited, Sydney
Prentice-Hall Canada Inc., Toronto
Prentice-Hall Hispanoamericana, S.A., Mexico
Prentice-Hall of India Private Limited, New Delhi
Prentice-Hall of Japan, Inc., Tokyo
Simon & Schuster Asia Pte. Ltd., Singapore
Editora Prentice-Hall do Brasil, Ltda., Rio de Janeiro

25

Applications: Electronic Mail (822, SMTP, MIME)

25.1 Introduction

This chapter continues our exploration of internetworking by considering electronic mail service and the protocols that support it. The chapter describes how a mail system is organized, explains alias expansion, and shows how mail system software uses the client-server paradigm to transfer each message.

25.2 Electronic Mail

Many users first encounter computer networks when they send or receive electronic mail (e-mail) to or from a remote site. E-mail is the most widely used application service. Indeed, many computer users access networks only through electronic mail.

E-mail is popular because it offers a fast, convenient method of transferring information. E-mail can accommodate small notes or large voluminous memos with a single mechanism. It should not surprise you to learn that more users send files with electronic mail than with file transfer protocols.

Mail delivery is a new concept because it differs fundamentally from other uses of networks that we have discussed. In all our examples, network protocols send packets directly to destinations, using timeout and retransmission for individual segments if no acknowledgement returns. In the case of electronic mail, however, the system must provide for instances when the remote machine or the network connections have failed. A sender does not want to wait for the remote machine to become available before continuing work, nor does the user want the transfer to abort merely because communication with the remote machine becomes temporarily unavailable.

To handle delayed delivery, mail systems use a technique known as *spooling*. When the user sends a mail message, the system places a copy in its private storage (spool†) area along with identification of the sender, recipient, destination machine, and time of deposit. The system then initiates the transfer to the remote machine as a background activity, allowing the sender to proceed with other computational activities. Figure 25.1 illustrates the idea.

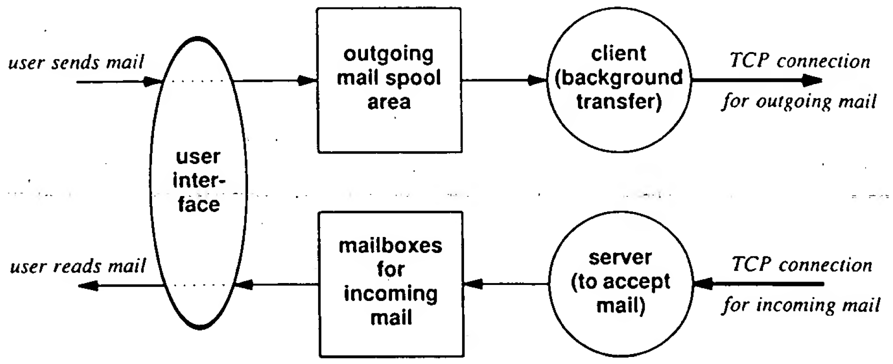
user sends mail
user reads mail
user inter- face
outgoing mail spool area
mailboxes for incoming mail
client (background transfer)
server (to accept mail)
TCP connection for outgoing mail
TCP connection for incoming mail

Figure 25.1 Conceptual components of an electronic mail system. The user invokes a user interface to deposit or retrieve mail; all transfers occur in the background.

The background mail transfer process becomes a client. The process first uses the domain name system to map the destination machine name to an IP address, and then attempts to form a TCP connection to the mail server on the destination machine. If it succeeds, the transfer process passes a copy of the message to the remote server, which stores the copy in the remote system's spool area. Once the client and server agree that the copy has been accepted and stored, the client removes the local copy. If it cannot form a TCP connection or if the connection fails, the transfer process records the time delivery was attempted and terminates. The background transfer process sweeps through the spool area periodically, typically once every 30 minutes, checking for undelivered mail. Whenever it finds a message or whenever a user deposits new outgoing mail, the background process attempts delivery again. If it finds that a mail message cannot be delivered after an extended time (e.g., 3 days), the mail software returns the message to the sender.

†Mail spool areas are sometimes called *mail queue* areas even though the term is technically inaccurate.

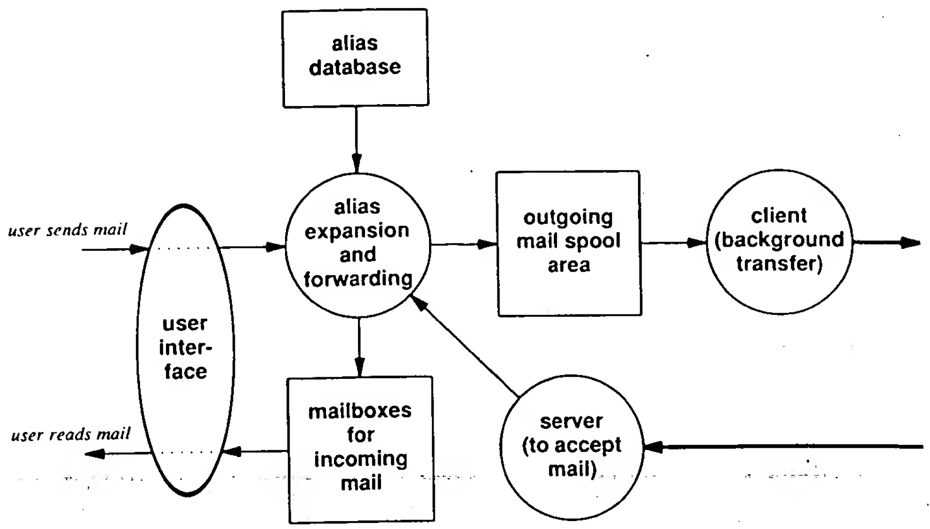

Figure 25.2 An extension of the mail system in Figure 25.1 that supports mail aliases and forwarding. Both incoming and outgoing mail passes through the alias expansion mechanism.

As Figure 25.2 shows, incoming and outgoing mail passes through the mail forwarder that expands aliases. Thus, if the alias database specifies that mail address *x* maps to replacement *y*, alias expansion will rewrite destination address *x*, changing it to *y*. The alias expansion program then determines whether *y* specifies a local or remote address, so it knows whether to place the message in the incoming mail queue or outgoing mail queue.

Mail alias expansion can be dangerous. Suppose two sites establish conflicting aliases. For example, assume site *A* maps mail address *x* into mail address *y* at site *B*, while site *B* maps mail address *y* into address *x* at site *A*. A mail message sent to address *x* at site *A* could bounce forever between the two sites†. Similarly, if the manager at site *A* accidentally maps a user's login name at that site to an address at another site, the user will be unable to receive mail. The mail may go to another user or, if the alias specifies an illegal address, senders will receive error messages.

25.5 The Relationship Of Internetworking And Mail

Many commercial computer systems can forward electronic mail from sites that do not connect to the Internet. How do such systems differ from the mail system described here? There are two crucial differences. First, a TCP/IP internet makes possible universal delivery service. Second, electronic mail systems built on TCP/IP are in-

†In practice, most mail forwarders terminate messages after the number of exchanges reaches a predetermined threshold.

herently more reliable than those built from arbitrary networks. The first idea is easy to understand. TCP/IP makes possible universal mail delivery because it provides universal interconnection among machines. In essence, all machines attached to an internet behave as if attached to a single, vendor independent network. With the basic network services in place, devising a standard mail exchange protocol becomes easier.

The second claim, that using TCP/IP makes mail delivery more reliable than other mechanisms, needs explanation. The key idea here is that TCP provides end-to-end connectivity. That is, mail software on the sending machine acts as a client, contacting a server on the ultimate destination. Only after the client successfully transfers a mail message to the server does it remove the message from the local machine. Thus, direct, end-to-end delivery enforces the following principle:

> *Mail systems that use end-to-end delivery can guarantee that each mail message remains in the sender's machine until it has been successfully copied to the recipient's machine.*

With such systems, the sender can always determine the exact status of a message by checking the local mail spool area.

The alternative form of electronic mail delivery uses *mail gateways*†, sometimes called *mail bridges*, *mail relays*, or *intermediate mail stops* to transfer messages. In such systems, the sender's machine does not contact the recipient's machine directly, but sends mail across one or more intermediate machines that forward it on.

The main disadvantage of using mail gateways is that they introduce unreliability. Once the sender's machine transfers a message to the first intermediate machine, it discards the local copy. Thus, while the message is in transit, neither the sender nor the recipient have a copy. Failures at intermediate machines may result in message loss without informing either the sender or recipient. Message loss can also result if the mail gateways route mail incorrectly. Another disadvantage of mail gateways is that they introduce delay. A mail gateway can hold messages for minutes, hours, or even days if it cannot forward them on to the next machine. Neither the sender nor receiver can determine where a message has been delayed, why it has not arrived, or how long the delay will last. The important point is that the sender and recipient must depend on machines over which they may have no control.

If mail gateways are less reliable than end-to-end delivery, why are they used? The chief advantage of mail gateways is interoperability. Mail gateways provide connections among standard TCP/IP mail systems and other mail systems, as well as between TCP/IP internets and networks that do not support Internet protocols. Suppose, for example, that company *X* has a large internal network and that employees use electronic mail, but that the network software does not support TCP/IP. Although it may be infeasible to make the company's network part of the connected Internet, it might be easy to place a mail gateway between the company's private network and the Internet, and to devise software that accepts mail messages from the local network and forwards them to the Internet.

†Readers should not confuse the term *mail gateway* with the term *IP gateway*, discussed earlier.

While the idea of mail gateways may seem somewhat awkward, electronic mail has turned into such an important tool that users who do not have Internet access depend on them. Thus, although gateways service is not as reliable or convenient as end-to-end delivery, it can still be useful.

25.6 TCP/IP Standards For Electronic Mail Service

Recall that the goal of the TCP/IP protocol effort is to provide for interoperability across the widest range of computer systems and networks. To extend the interoperability of electronic mail, TCP/IP divides its mail standards into two sets. One standard specifies the format for mail messages†. The other specifies the details of electronic mail exchange between two computers. Keeping the two standards for electronic mail separate makes it possible to build mail gateways that connect TCP/IP internets to some other vendor's mail delivery system, while still using the same message format for both.

As anyone who has used electronic mail knows, each memo is divided into two parts: a header and a body, separated by a blank line. The TCP/IP standard for mail messages specifies the exact format of mail headers as well as the semantic interpretation of each header field; it leaves the format of the body up to the sender. In particular, the standard specifies that headers contain readable text, divided into lines that consist of a keyword followed by a colon followed by a value. Some keywords are required, others are optional, and the rest are uninterpreted. For example, the header must contain a line that specifies the destination. The line begins *To:* and contains the electronic mail address of the intended recipient on the remainder of the line. A line that begins *From:* contains the electronic mail address of the sender. Optionally, the sender may specify an address to which replies should be sent (i.e., to allow the sender to specify that replies should be sent to an address other than the sender's mailbox). If present, a line that begins *Reply-to:* specifies the address for replies. If no such line exists, the recipient will use information on the *From:* line as the return address.

The mail message format is chosen to make it easy to process and transport across heterogeneous machines. Keeping the mail header format straightforward allows it to be used on a wide range of systems. Restricting messages to readable text avoids the problems of selecting a standard binary representation and translating between the standard representation and the local machine's representation.

25.7 Electronic Mail Addresses

A user familiar with electronic mail knows that mail address formats vary among e-mail systems. Thus, it can be difficult to determine a correct electronic mail address, or even to understand the sender's intentions. Within the global Internet, addresses have a simple, easy to remember form:

local-part @ domain-name

†Mail system experts often refer to the mail message format as "822" because RFC 822 contains the standard (RFC 733 is a former standard no longer used).

where *domain-name* is the domain name of a mail destination† to which the mail should be delivered, and *local-part* is the address of a mailbox on that machine. For example, within the Internet, the author's electronic mail address is:

comer @ purdue . edu

However, mail gateways make addresses complex. Someone outside the Internet must either address the mail to the nearest mail gateway or have software that automatically does so. For example, when CSNET operated a mail gateway that connected between outside networks and the Internet, someone with access to the gateway might have used the following address to reach the author:

comer % purdue . edu @ relay . cs . net

Once the mail reached machine *relay.cs.net*, the mail gateway software extracted *local-part*, changed the percent sign (%) into an at sign (@), and used the result as a destination address to forward the mail.

The reason addresses become complex when they include non-Internet sites is that the electronic mail address mapping function is local to each machine. Thus, some mail gateways require the local part to contain addresses of the form:

user % domain-name

while others require:

user : domain-name

and still others use completely different forms. More important, electronic mail systems do not usually agree on conventions for precedence or quoting, making it impossible for a user to guarantee how addresses will be treated. For example, consider the electronic mail address:

comer % purdue . edu @ relay . cs . net

mentioned earlier. A site using the TCP/IP standard for mail would interpret the address to mean, "send the message to mail exchanger *relay.cs.net* and let that mail exchanger decide how to interpret *comer % purdue.edu*" (the local part). In essence, the sites act as if the address were parenthesized:

(comer % purdue . edu) @ (relay . cs . net)

At sites that use % to separate user names from destination machines, the same address might mean, "send the mail to user *comer* at the site given by the remainder of the address." That is, such sites act as if the address were parenthesized:

(comer) % (purdue . edu @ relay . cs . net)

†Technically, the domain name specifies a *mail exchanger*, not a machine name.

We can summarize the problem:

> *Because each mail gateway determines the exact details of how it interprets and maps electronic mail addresses, there is no standard for addresses that cross mail gateway boundaries to networks outside the Internet.*

25.8 Pseudo Domain Addresses

To help solve the problem of multiple mail systems, each with its own e-mail address format, a site can use domain-style names for all e-mail addresses, even if the site does not use the domain name system. For example, a site that uses UUCP can implement a pseudo-domain, *uucp*, that allows users to specify mail addresses of the form:

uucp-style address @ uucp

or a related form:

user @ uucp-site . uucp

The local mail forwarding software recognizes the special addresses and translates them to the address syntax required by the UUCP network software. From the user's perspective, the advantage is clear: all electronic addresses have the same general format independent of the underlying communication network used to reach the recipient. Of course, such addresses only work where local mailers have been instructed to map them into appropriate forms and only when the appropriate transport mechanisms are available. Furthermore, even though pseudo-domain mail addresses have the same form as domain names, they can only be used with electronic mail – one cannot find IP addresses or mail exchanger addresses for them using the domain name system.

25.9 Simple Mail Transfer Protocol (SMTP)

In addition to message formats, the TCP/IP protocol suite specifies a standard for the exchange of mail between machines. That is, the standard specifies the exact format of messages a client on one machine uses to transfer mail to a server on another. The standard transfer protocol is known as *SMTP*, the *Simple Mail Transfer Protocol*. As you might guess, SMTP is simpler than an earlier *Mail Transfer Protocol, MTP*. The SMTP protocol focuses specifically on how the underlying mail delivery system passes messages across a link from one machine to another. It does not specify how the mail system accepts mail from a user or how the user interface presents the user with incoming mail. Also, SMTP does not specify how mail is stored or how frequently the mail system attempts to send messages.

SMTP is surprisingly straightforward. Communication between a client and server consists of readable ASCII text. Although SMTP rigidly defines the command format, humans can easily read a transcript of interactions between a client and server. Initially, the client establishes a reliable stream connection to the server and waits for the server to send a *220 READY FOR MAIL* message. (If the server is overloaded, it may delay sending the *220* message temporarily.) Upon receipt of the *220* message, the client sends a *HELO*† command. The end of a line marks the end of a command. The server responds by identifying itself. Once communication has been established, the sender can transmit one or more mail messages, terminate the connection, or request the server to exchange the roles of sender and receiver so messages can flow in the opposite direction. The receiver must acknowledge each message. It can also abort the entire connection or abort the current message transfer.

Mail transactions begin with a *MAIL* command that gives the sender identification as well as a *FROM:* field that contains the address to which errors should be reported. A recipient prepares its data structures to receive a new mail message, and replies to a *MAIL* command by sending the response *250*. Response *250* means that all is well. The full response consists of the text *250 OK*. As with other application protocols, programs read the abbreviated commands and 3-digit numbers at the beginning of lines; the remaining text is intended to help humans debug mail software.

After a successful *MAIL* command, the sender issues a series of *RCPT* commands that identify recipients of the mail message. The receiver must acknowledge each *RCPT* command by sending *250 OK* or by sending the error message *550 No such user here*.

After all *RCPT* commands have been acknowledged, the sender issues a *DATA* command. In essence, a *DATA* command informs the receiver that the sender is ready to transfer a complete mail message. The receiver responds with message *354 Start mail input* and specifies the sequence of characters used to terminate the mail message. The termination sequence consists of 5 characters: carriage return, line feed, period, carriage return, and line feed‡.

An example will clarify the SMTP exchange. Suppose user *Smith* at host *Alpha.EDU* sends a message to users *Jones*, *Green*, and *Brown* at host *Beta.GOV*. The SMTP client software on host *Alpha.EDU* contacts the SMTP server software on host *Beta.GOV* and begins the exchange shown in Figure 25.3.

†*HELO* is an abbreviation for "hello."

‡SMTP uses *CR LF* to terminate a line, and forbids the body of a mail message to have a period on a line by itself.

```
S: 220 Beta.GOV Simple Mail Transfer Service Ready
C: HELO Alpha.EDU
S: 250 Beta.GOV

C: MAIL FROM:<Smith@Alpha.EDU>
S: 250 OK

C: RCPT TO:<Jones@Beta.GOV>
S: 250 OK

C: RCPT TO:<Green@Beta.GOV>
S: 550 No such user here

C: RCPT TO:<Brown@Beta.GOV>
S: 250 OK

C: DATA
S: 354 Start mail input; end with <CR><LF>.<CR><LF>
C: ...sends body of mail message...
C: ...continues for as many lines as message contains
C: <CR><LF>.<CR><LF>
S: 250 OK

C: QUIT
S: 221 Beta.GOV Service closing transmission channel
```

Figure 25.3 Example of SMTP transfer from Alpha.EDU to Beta.GOV. Lines that begin with "C:" are transmitted by the client (Alpha), while lines that begin "S:" are transmitted by the server. In the example, machine Beta.GOV does not recognize the intended recipient Green.

In the example, the server rejects recipient *Green* because it does not recognize the name as a valid mail destination (i.e., it is neither a user nor a mailing list). The SMTP protocol does not specify the details of how a client handles such errors – the client must decide. Although clients can abort the delivery completely if an error occurs, most clients do not. Instead, they continue delivery to all valid recipients and then report problems to the original sender. Usually, the client reports errors using electronic mail. The error message contains a summary of the error as well as the header of the mail message that caused the problem.

Once a client has finished sending all the mail messages it has for a particular destination, the client may issue the *TURN*† command to turn the connection around. If it does, the receiver responds *250 OK* and assumes control of the connection. With the roles reversed, the side that was originally a server sends back any waiting mail mes-

†In practice, few mail servers use the *TURN* command.

sages. Whichever side controls the interaction can choose to terminate the session. To do so, it issues a *QUIT* command. The other side responds with command *221*, which means it agrees to terminate. Both sides then close the TCP connection gracefully.

SMTP is much more complex than we have outlined here. For example, if a user has moved, the server may know the user's new mailbox address. SMTP allows the server to choose to inform the client about the new address so the client can use it in the future. When informing the client about a new address, the server may choose to forward the mail that triggered the message, or it may request that the client take the responsibility for forwarding.

25.10 The MIME Extension For Non-ASCII Data

To allow transmission of non-ASCII data through e-mail, the IETF defined the *Multipurpose Internet Mail Extensions* (*MIME*). MIME does not change SMTP or replace it. Instead, MIME allows arbitrary data to be encoded in ASCII and then transmitted in a standard e-mail message. To accommodate arbitrary data types and representations, each MIME message includes information that tells the recipient the type of the data and the encoding used. MIME information resides in the 822 mail header – the MIME header lines specify the version of MIME used, the type of the data being sent, and the encoding used to convert the data to ASCII. For example, Figure 25.4 illustrates a MIME message that contains a photograph in standard *GIF*† representation. The GIF image has been converted to a 7-bit ASCII representation using the *base64* encoding.

```
From: bill@acollege.edu
To: john@somewhere.com
MIME-Version: 1.0
Content-Type: image/gif
Content-Transfer-Encoding: base64

        ...data for the image...
```

Figure 25.4 An example MIME message. Lines in the header identify the type of the data as well as the encoding used.

In the figure, the header line *MIME-Version:* declares that the message was composed using version *1.0* of the MIME protocol. The *Content-Type:* declaration specifies that the data is a GIF image, and the *Content-Transfer-Encoding:* header declares that *base64* encoding was used to convert the image to ASCII. To view the image, a receiver's mail system must first convert from *base64* encoding back to binary, and then run an application that displays a GIF image on the user's screen.

The MIME standard specifies that a *Content-Type* declaration must contain two identifiers, a *content type* and a *subtype*, separated by a slash. In the example, *image* is the content type, and *gif* is the subtype.

†GIF is the Graphics Interchange Format.

The standard defines seven basic content types, the valid subtypes for each, and transfer encodings. For example, although an *image* must be of subtype *jpeg* or *gif*, *text* cannot use either subtype. In addition to the standard types and subtypes, MIME permits a sender and receiver to define private content types†. Figure 25.5 lists the seven basic content types.

Content Type	Used When Data In the Message Is
text	Textual (e.g. a document).
image	A still photograph or computer-generated image
audio	A sound recording
video	A video recording that includes motion
application	Raw data for a program
multipart	Multiple messages that each have a separate content type and encoding
message	An entire e-mail message (e.g., a memo that has been forwarded) or an external reference to a message (e.g., an FTP server and file name)

Figure 25.5 The seven basic types that can appear in a MIME *Content-Type* declaration and their meanings.

25.11 MIME Multipart Messages

The MIME multipart content type is useful because it adds considerable flexibility. The standard defines four possible subtypes for a multipart message; each provides important functionality. Subtype *mixed* allows a single message to contain multiple, independent submessages that each can have an independent type and encoding. Mixed multipart messages make it possible to include text, graphics, and audio in a single message, or to send a memo with additional data segments attached, similar to *enclosures* included with a business letter. Subtype *alternative* allows a single message to include multiple representations of the same data. Alternative multipart messages are useful when sending a memo to many recipients who do not all use the same hardware and software system. For example, one can send a document as both plain ASCII text and in formatted form, allowing recipients who have computers with graphic capabilities to select the formatted form for viewing. Subtype *parallel* permits a single message to include subparts that should be viewed together (e.g., video and audio subparts that must be played simultaneously). Finally, subtype *digest* permits a single message to contain a set of other messages (e.g., a collection of the e-mail messages from a discussion).

Figure 25.6 illustrates one of the prime uses for multipart messages: an e-mail message can contain both a short text that explains the purpose of the message and other parts that contain nontextual information. In the figure, a note in the first part of the message explains that the second part contains a photographic image.

†To avoid potential name conflicts, the standard requires that names chosen for private content types each begin with the string *X-*.

```
From: bill@acollege.edu
To: john@somewhere.com
MIME-Version: 1.0
Content-Type: Multipart/Mixed; Boundary=StartOfNextPart

--StartOfNextPart
John,
   Here is the photo of our research lab that I promised
to send you.  You can see the equipment you donated.

Thanks again,
Bill

--StartOfNextPart
Content-Type: image/gif
Content-Transfer-Encoding: base64
     ...data for the image...
```

Figure 25.6 An example of a MIME mixed multipart message. Each part of the message can have an independent content type.

The figure also illustrates a few details of MIME. For example, each header line can contain parameters of the form $X = Y$ after basic declarations. The keyword *Boundary=* following the multipart content type declaration in the header defines the string used to separate parts of the message. In the example, the sender has selected the string *StartOfNextPart* to serve as the boundary. Declarations of the content type and transfer encoding for a submessage, if included, immediately follow the boundary line. In the example, the second submessage is declared to be a GIF image.

25.12 Summary

Electronic mail is among the most widely available application services. Like most TCP/IP services, it uses the client-server paradigm. The mail system buffers outgoing and incoming messages, allowing the transfer from client and server to occur in background.

The TCP/IP protocol suite provides separate standards for mail message format and mail transfer. The mail message format, called *822*, uses a blank line to separate a message header and the body. The Simple Mail Transfer Protocol (SMTP) defines how a mail system on one machine transfers mail to a server on another.

The Multipurpose Internet Mail Extensions (MIME) provides a mechanism that allows arbitrary data to be transferred using SMTP. MIME adds lines to the header of an e-mail message to define the type of the data and the encoding used. MIME's mixed multipart type permits a single message to contain multiple data types.

FOR FURTHER STUDY

The protocols described in this chapter are all specified in Internet RFCs. Postel [RFC 821] describes the Simple Mail Transfer Protocol and gives many examples. The exact format of mail messages is given by Crocker [RFC 822]. Borenstein and Freed [RFC 1521] specifies the standard for MIME, including the syntax of header declarations, the interpretation of content types, and the *base64* encoding mentioned in this chapter. Moore [RFC 1522] defines MIME header extensions for non-ASCII text, and Postel [RFC 1590] describes the procedure for registration of new content and encoding types. Partridge [RFC 974] discusses the relationship between mail routing and the domain name system. Horton [RFC 976] proposes a standard for the UNIX UUCP mail system.

EXERCISES

25.1 Some mail systems force the user to specify a sequence of machines through which the message should travel to reach its destination. The mail protocol in each machine merely passes the message on to the next machine. List three disadvantages of such a scheme.

25.2 Find out if your computing system allows you to invoke SMTP directly.

25.3 Build an SMTP client and use it to deliver a mail message.

25.4 See if you can send mail through a mail gateway and back to yourself.

25.5 Make a list of mail address forms that your site handles and write a set of rules for parsing them.

25.6 Find out how the UNIX *sendmail* program can be used to implement a mail gateway.

25.7 Find out how often your local mail system attempts delivery and how long it will continue before giving up.

25.8 Many mail systems allow users to direct incoming mail to a program instead of storing it in a mailbox. Build a program that accepts your incoming mail, places your mail in a file, and then sends a reply to tell the sender you are on vacation.

25.9 Read the SMTP standard carefully. Then use TELNET to connect to the SMTP port on a remote machine and ask the remote SMTP server to expand a mail alias.

25.10 A user receives mail in which the *To* field specifies the string *important-people*. The mail was sent from a computer on which the alias *important-people* includes no valid mailbox identifiers. Read the SMTP specification carefully to see how such a situation is possible.

25.11 Read the MIME standard carefully. What servers can be specified in a MIME external reference?

Appendix B

SN 08/738,659

“TCP/IP Illustrated,” Vol. 1 W. Richard Stevens 1994, Addison-Wesley, pp. 441-452.

28

SMTP: Simple Mail Transfer Protocol

Introduction

Electronic mail (e-mail) is undoubtedly one of the most popular applications. [Caceres et al. 1991] shows that about one-half of all TCP connections are for the *Simple Mail Transfer Protocol*, SMTP. (On a byte count basis, FTP connections carry more data.) [Paxson 1993] found that the average mail message contains around 1500 bytes of data, but some messages contain megabytes of data, because electronic mail is sometimes used to send files.

Figure 28.1 shows an outline of e-mail exchange using TCP/IP.

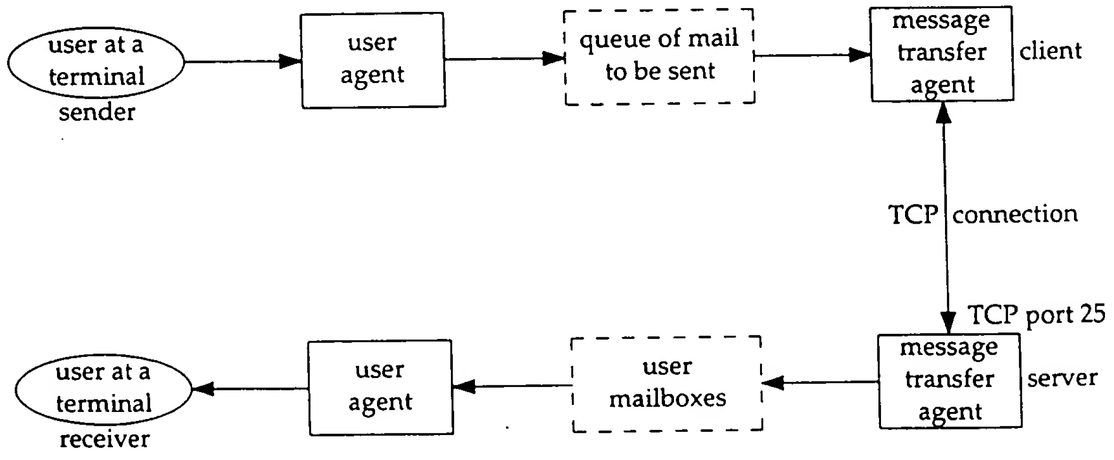

Figure 28.1 Outline of Internet electronic mail.

Users deal with a *user agent*, of which there are a multitude to choose from. Popular user agents for Unix include MH, Berkeley Mail, Elm, and Mush.

The exchange of mail using TCP is performed by a *message transfer agent* (MTA). The most common MTA for Unix systems is Sendmail. Users normally don't deal with the MTA. It is the responsibility of the system administrator to set up the local MTA. Users often have a choice, however, for their user agent.

This chapter examines the exchange of electronic mail between the two MTAs using TCP. We do not look at the operation or design of user agents.

RFC 821 [Postel 1982] specifies the SMTP protocol. This is how two MTAs communicate with each other across a single TCP connection. RFC 822 [Crocker 1982] specifies the format of the electronic mail message that is transmitted using RFC 821 between the two MTAs.

28.2 SMTP Protocol

The communication between the two MTAs uses NVT ASCII. Commands are sent by the client to the server, and the server responds with numeric reply codes and optional human-readable strings. This is similar to what we saw with FTP in the previous chapter.

There are a small number of commands that the client can send to the server: less than a dozen. (By comparison, FTP has more than 40 commands.) Rather than describing each one, we'll start with a simple example to show what happens when we send mail.

Simple Example

We'll send a simple one-line message and watch the SMTP connection. We invoke our user agent with the -v flag, which is passed to the mail transport agent (Sendmail in this case). This MTA displays what is sent and received across the SMTP connection when this flag is specified. Lines beginning with >>> are commands sent by the SMTP client, and lines beginning with a 3-digit reply code are from the SMTP server. Here is the interactive session:

```
sun % mail -v rstevens@noao.edu          invoke our user agent
To: rstevens@noao.edu                    this is output by user agent
Subject: testing                         we're then prompted for a subject
                                         user agent adds one blank line between headers and body
1, 2, 3.                                 this is what we type as the body of the message
.                                        we type a period on a line by itself to say we're done

Sending letter ... rstevens@noao.edu...     verbose output from user agent
                                            following is output by MTA (Sendmail)
Connecting to mailhost via ether...
Trying 140.252.1.54...  connected.
220 noao.edu Sendmail 4.1/SAG-Noao.G89 ready at Mon, 19 Jul 93 12:47:34 MST

>>> HELO sun.tuc.noao.edu.
250 noao.edu Hello sun.tuc.noao.edu., pleased to meet you

>>> MAIL From:<rstevens@sun.tuc.noao.edu>
250 <rstevens@sun.tuc.noao.edu>... Sender ok
```

```
>>> RCPT To:<rstevens@noao.edu>
250 <rstevens@noao.edu>... Recipient ok

>>> DATA
354 Enter mail, end with "." on a line by itself

>>> .
250 Mail accepted

>>> QUIT
221 noao.edu delivering mail

rstevens@noao.edu... Sent
sent.                                   this is output by user agent
```

Only five SMTP commands are used to send the mail: HELO, MAIL, RCPT, DATA, and QUIT.

We type `mail` to invoke our user agent. We're then prompted for a subject, and after typing that, we type the body of the message. Typing a period on a line by itself completes the message and the user agent passes the mail to the MTA for delivery.

The client does the active open to TCP port 25. When this returns, the client waits for a greeting message (reply code 220) from the server. This server's response must start with the fully qualified domain name of the server's host: `noao.edu` in this example. (Normally the text that follows the numeric reply code is optional. Here the domain name is required. The text beginning with `Sendmail` is optional.)

Next the client identifies itself with the HELO command. The argument must be the fully qualified domain name of the client host: `sun.tuc.noao.edu`.

The MAIL command identifies the originator of the message. The next command, RCPT, identifies the recipient. More than one RCPT command can be issued if there are multiple recipients.

The contents of the mail message are sent by the client using the DATA command. The end of the message is specified by the client sending a line containing just a period. The final command, QUIT, terminates the mail exchange.

Figure 28.2 is a time line of the SMTP connection between the sender SMTP (the client) and the receiver SMTP (the server). We have removed the connection establishment and termination, and the window size advertisements.

The amount of data we typed to our user agent was a one-line message ("1, 2, 3."), yet 393 bytes of data are sent in segment 12. The following 12 lines comprise the 393 bytes that are sent by the client:

```
Received: by sun.tuc.noao.edu. (4.1/SMI-4.1)
        id AA00502; Mon, 19 Jul 93 12:47:32 MST
Message-Id: <9307191947.AA00502@sun.tuc.noao.edu.>
From: rstevens@sun.tuc.noao.edu (Richard Stevens)
Date: Mon, 19 Jul 1993 12:47:31 -0700
Reply-To: rstevens@noao.edu
X-Phone: +1 602 676 1676
X-Mailer: Mail User's Shell (7.2.5 10/14/92)
To: rstevens@noao.edu
Subject: testing

1, 2, 3.
```

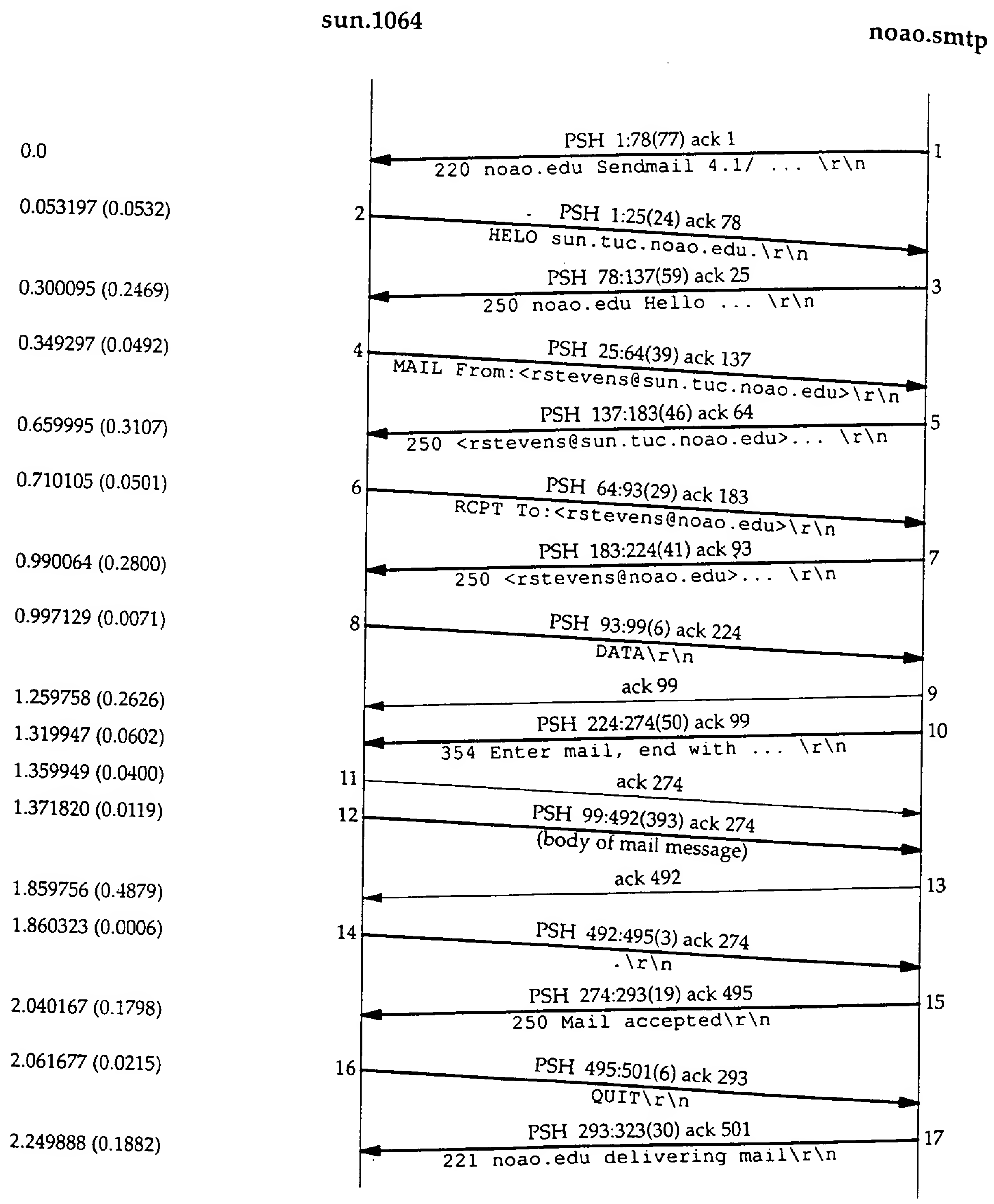

Figure 28.2 Basic SMTP mail delivery.

The first three lines, `Received:` and `Message-Id:`, are added by the MTA, and the

SMTP Commands

The minimal SMTP implementation supports eight commands. We saw five of them in the previous example: HELO, MAIL, RCPT, DATA, and QUIT.

The RSET command aborts the current mail transaction and causes both ends to reset. Any stored information about sender, recipients, or mail data is discarded.

The VRFY command lets the client ask the sender to verify a recipient address, without sending mail to the recipient. It's often used by a system administrator, by hand, for debugging mail delivery problems. We'll show an example of this in the next section.

The NOOP command does nothing besides force the server to respond with an OK reply code (200).

There are additional, optional commands. EXPN expands a mailing list, and is often used by the system administrator, similar to VRFY. Indeed, most versions of Sendmail handle the two identically.

Version 8 of Sendmail in 4.4BSD no longer handles the two identically. VRFY does not expand aliases and doesn't follow `.forward` files.

The TURN command lets the client and server switch roles, to send mail in the reverse direction, without having to take down the TCP connection and create a new one. (Sendmail does not support this command.) There are three other commands (SEND, SOML, and SAML), which are rarely implemented, that replace the MAIL command. These three allow combinations of the mail being delivered directly to the user's terminal (if logged in), or sent to the recipient's mailbox.

Envelopes, Headers, and Body

Electronic mail is composed of three pieces.

1. The *envelope* is used by the MTAs for delivery. In our example the envelope was specified by the two SMTP commands:

   ```
   MAIL From:<rstevens@sun.tuc.noao.edu>
   RCPT To:<rstevens@noao.edu>
   ```

   RFC 821 specifies the contents and interpretation of the envelope, and the protocol used to exchange mail across a TCP connection.

2. *Headers* are used by the user agents. We saw nine header fields in our example: `Received`, `Message-Id`, `From`, `Date`, `Reply-To`, `X-Phone`, `X-Mailer`, `To`, and `Subject`. Each header field contains a name, followed by a colon, followed by the field value. RFC 822 specifies the format and interpretation of the header fields. (Headers beginning with an `X-` are user-defined fields. The others are defined by RFC 822.) Long header fields, such as `Received` in the example, are folded onto multiple lines, with the additional lines starting with white space.

3. The *body* is the content of the message from the sending user to the receiving user. RFC 822 specifies the body as lines of NVT ASCII text. When transferred

using the DATA command, the headers are sent first, followed by a blank line, followed by the body. Each line transferred using the DATA command must be less than 1000 bytes.

The user agent takes what we specify as the body, adds some headers, and passes the result to the MTA. The MTA adds a few headers, adds the envelope, and sends the result to another MTA.

The term *content* is often used to describe the combination of headers and the body. The content is sent by the client with the DATA command.

elay Agents

The first line of informational output by our local MTA in our example is "Connecting to mailhost via ether." This is because the author's system has been configured to send all nonlocal outgoing mail to a relay machine for delivery.

This is done for two reasons. First, it simplifies the configuration of all MTAs other than the relay system's MTA. (Configuring an MTA is not simple, as anyone who has ever worked with Sendmail can attest to.) Second, it allows one system at an organization to act as the mail hub, possibly hiding all the individual systems.

In this example the relay system has a hostname of `mailhost` in the local domain (`.tuc.noao.edu`) and all the individual systems are configured to send their mail to this host. We can execute the `host` command to see how this name is defined to the DNS:

```
sun % host mailhost
mailhost.tuc.noao.edu   CNAME   noao.edu        canonical name
noao.edu                A       140.252.1.54    its real IP address
```

If the host used as the relay changes in the future, only its DNS name need change—the mail configuration of all the individual systems does not change.

Most organizations are using relay systems today. Figure 28.3 is a revised picture of Internet mail (Figure 28.2), taking into account that both the sending host and the final receiving host probably use a relay host.

In this scenario there are four MTAs between the sender and receiver. The local MTA on the sender's host just delivers the mail to its relay MTA. (This relay MTA could have a hostname of `mailhost` in the organization's domain.) This communication uses SMTP across the organization's local internet. The relay MTA in the sender's organization then sends the mail to the receiving organization's relay MTA across the Internet. This other relay MTA then delivers the mail to the receiver's host, by communication with the local MTA on the receiver's host. All the MTAs in this example use SMTP, although the possibility exists for other protocols to be used.

IVT ASCII

One feature of SMTP is that it uses NVT ASCII for everything: the envelope, the headers, and the body. As we said in Section 26.4, this is a 7-bit character code, transmitted as 8-bit bytes, with the high-order bit set to 0.

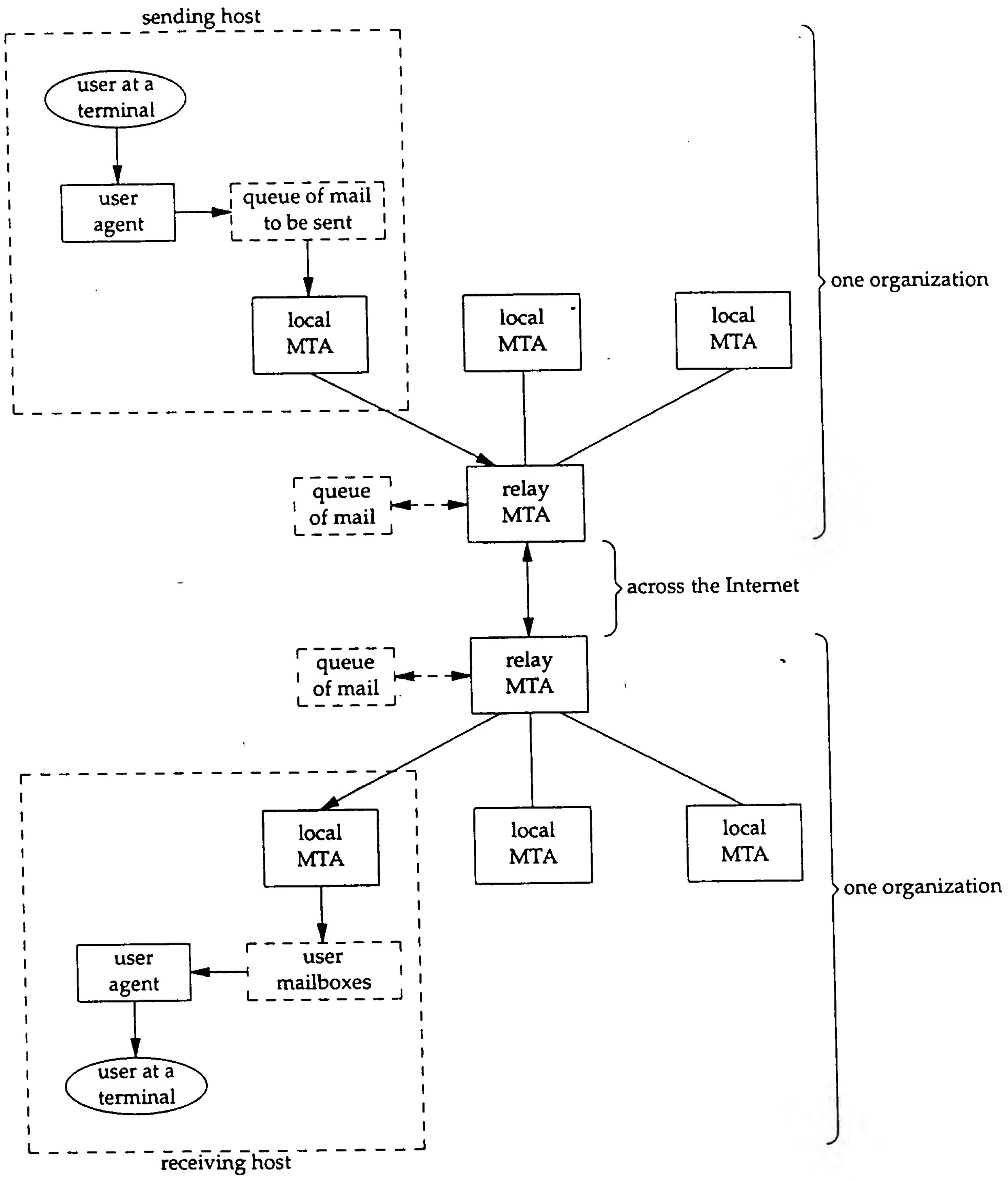

Figure 28.3 Internet electronic mail, with a relay system at both ends.

In Section 28.4 we discuss some newer features of Internet mail, extended SMTP and multimedia mail (MIME), that allow the sending and receiving of data such as audio and video. We'll see that MIME works with NVT ASCII for the envelope, headers, and body, with changes required only in the user agents.

Retry Intervals

When a user agent passes a new mail message to its MTA, delivery is normally attempted immediately. If the delivery fails, the MTA must queue the message and try again later.

The Host Requirements RFC recommends an initial timeout of at least 30 minutes. The sender should not give up for at least 4–5 days. Furthermore, since delivery failures are often transient (the recipient has crashed or there is a temporary loss of network connectivity), it makes sense to try two connection attempts during the first hour that the message is in the queue.

28.3 SMTP Examples

We showed normal mail delivery in the previous section, so here we'll show how MX records are used for mail delivery, and illustrate the VRFY and EXPN commands.

MX Records: Hosts Not Directly Connected to the Internet

In Section 14.6 we mentioned that one type of resource record in the DNS is the mail exchange record, called MX records. In the following example we'll show how MX records are used to send mail to hosts that are not directly connected to the Internet. RFC 974 [Partridge 1986] describes the handling of MX records by MTAs.

The host `mlfarm.com` is not directly connected to the Internet, but has an MX record that points to a mail forwarder that is on the Internet:

```
sun % host -a -v -t mx mlfarm.com
The following answer is not authoritative:
mlfarm.com              86388    IN    MX    10 mercury.hsi.com
mlfarm.com              86388    IN    MX    15 hsi86.hsi.com
Additional information:
mercury.hsi.com         86388    IN     A    143.122.1.91
hsi86.hsi.com          172762    IN     A    143.122.1.6
```

There are two MX records, each with a different preference. We expect the MTA to start with the lower of the two preference values.

The following script shows mail being sent to this host:

```
sun % mail -v ron@mlfarm.com              -v flag to see what the MTA does
To: ron@mlfarm.com
Subject: MX test message

                                          the body of the message is typed here (not shown)
.                                         period on a line by itself to terminate message

Sending letter ... ron@mlfarm.com...
Connecting to mlfarm.com via tcp...
mail exchanger is mercury.hsi.com         the MX records are found
Trying 143.122.1.91... connected.         first tries the one with lower preference
220 mercury.hsi.com ...
                                          remainder is normal SMTP mail transfer
```

We can see in this output that the MTA discovered that the destination host had an MX record and used the MX record with the lowest preference value.

Before running this example from the host sun, it was configured not to use its normal relay host, so we could see the mail exchange with the destination host. It was also configured to use the name server on the host noao.edu (which is across its dialup SLIP link), so we could capture both the mail transfer and the DNS traffic using tcpdump on the SLIP link. Figure 28.4 shows the starting portion of the tcpdump output.

```
1  0.0                   sun.1624 > noao.edu.53: 2+ MX? mlfarm.com. (28)
2  0.445572 (0.4456)   noao.edu.53 > sun.1624: 2* 2/0/2 MX
                                                  mercury.hsi.com. 10 (113)

3  0.505739 (0.0602)   sun.1143 > mercury.hsi.com.25: S 1617536000:1617536000(0)
                                                         win 4096
4  0.985428 (0.4797)   mercury.hsi.com.25 > sun.1143: S 1832064000:1832064000(0)
                                                         ack 1617536001 win 16384
5  0.986003 (0.0006)   sun.1143 > mercury.hsi.com.25: . ack 1 win 4096
6  1.735360 (0.7494)   mercury.hsi.com.25 > sun.1143: P 1:90(89) ack 1 win 16384
```

Figure 28.4 Sending mail to a host that uses MX records.

In line 1 the MTA queries its name server for an MX record for mlfarm.com. The plus sign following the 2 means the recursion-desired flag is set. The response in line 2 has the authoritative bit set (the asterisk following the 2) and contains 2 answer RRs (the two MX host names), 0 authority RRs, and 2 additional RRs (the IP addresses of the two hosts).

In lines 3–5 a TCP connection is established with the SMTP server on the host mercury.hsi.com. The server's initial 220 response is shown in line 6.

Somehow the host mercury.hsi.com must deliver this mail message to the destination, mlfarm.com. The UUCP protocols are a popular way for a system not connected to the Internet to exchange mail with its MX site.

In this example the MTA asks for an MX record, gets a positive result, and sends the mail. Unfortunately the interaction between an MTA and the DNS can differ between implementations. RFC 974 specifies that an MTA should ask for MX records first, and if none are found, attempt delivery to the destination host (i.e., ask the DNS for an A record for the host, for its IP address). MTAs must also deal with CNAME records in the DNS (canonical names).

As an example, if we send mail to rstevens@mailhost.tuc.noao.edu from a BSD/386 host, the following steps are executed by the MTA (Sendmail).

1. Sendmail asks the DNS for CNAME records for mailhost.tuc.noao.edu. We see that a CNAME record exists:

   ```
   sun % host -t cname mailhost.tuc.noao.edu
   mailhost.tuc.noao.edu    CNAME    noao.edu
   ```

2. A DNS query is issued for CNAME records for noao.edu and the response says none exist.

3. Sendmail then asks the DNS for MX records for noao.edu and gets one MX record:

```
sun % host -t mx noao.edu
noao.edu                        MX         noao.edu
```

4. Sendmail queries the DNS for an A record (IP address) for noao.edu and gets back the value 140.252.1.54. (This A record was probably returned by the name server for noao.edu as an additional RR with the MX reply in step 3.)
5. An SMTP connection is initiated to 140.252.1.54 and the mail is sent.

A CNAME query is not tried for the data returned in the MX record (noao.edu). The data in the MX record cannot be an alias—it must be the name of a host that has an A record.

The version of Sendmail distributed with SunOS 4.1.3 that uses the DNS only queries for MX records, and gives up if an MX record isn't found.

MX Records: Hosts That Are Down

Another use of MX records is to provide an alternative mail receiver when the destination host is down. If we look at the DNS entry for our host sun we see that it has two MX records:

```
sun % host -a -v -t mx sun.tuc.noao.edu
sun.tuc.noao.edu       86400      IN    MX       0 sun.tuc.noao.edu
sun.tuc.noao.edu       86400      IN    MX      10 noao.edu
Additional information:
sun.tuc.noao.edu       86400      IN     A      140.252.1.29
sun.tuc.noao.edu       86400      IN     A      140.252.13.33
noao.edu               86400      IN     A      140.252.1.54
```

The MX record with the lowest preference indicates that direct delivery to the host itself should be tried first, and the next preference is to deliver the mail to the host noao.edu.

In the following script we send mail to ourself at the host sun.tuc.noao.edu, from the host vangogh.cs.berkeley.edu, after turning off the destination's SMTP server. When a connection request arrives for port 25, TCP should respond with an RST, since no process has a passive open pending for that port.

```
vangogh % mail -v rstevens@sun.tuc.noao.edu
A test to a host that's down.
.
EOT
rstevens@sun.tuc.noao.edu... Connecting to sun.tuc.noao.edu. (smtp)...
rstevens@sun.tuc.noao.edu... Connecting to noao.edu. (smtp)...
220 noao.edu ...
                                          remainder is normal SMTP mail transfer
```

We see that the MTA tries to contact sun.tuc.noao.edu and then gives up and contacts noao.edu instead.

Figure 28.5 is the tcpdump output that shows that TCP responds to the incoming SYNs with an RST.

```
1 0.0                 vangogh.3873 > 140.252.1.29.25: S 2358303745:2358303745(0) ...
2 0.000621 (0.0006) 140.252.1.29.25 > vangogh.3873: R 0:0(0) ack 2358303746 win 0

3 0.300203 (0.2996) vangogh.3874 > 140.252.13.33.25: S 2358367745:2358367745(0) ...
4 0.300620 (0.0004) 140.252.13.33.25 > vangogh.3874: R 0:0(0) ack 2358367746 win 0
```

Figure 28.5 Attempt to connect to an SMTP server that is not running.

In line 1 vangogh sends a SYN to port 25 at the primary IP address for sun: 140.252.1.29. This is rejected in line 2. The SMTP client on vangogh then tries the next IP address for sun: 140.252.13.33 (line 3), and it also causes an RST to be returned (line 4).

The SMTP client doesn't try to differentiate between the different error returns from its active open on line 1, which is why it tries the other IP address on line 2. If the error had been something like "host unreachable" for the first attempt, it's possible that the second attempt could work.

If the reason the SMTP client's active open fails is because the server host is down, we would see the client retransmit the SYN to IP address 140.252.1.29 for a total of 75 seconds (similar to Figure 18.6), followed by the client sending another three SYNs to IP address 140.252.13.33 for another 75 seconds. After 150 seconds the client would move on to the next MX record with the higher preference.

VRFY and EXPN Commands

The VRFY command verifies that a recipient address is OK, without actually sending mail. EXPN is intended to expand a mailing list, without sending mail to the list. Many SMTP implementations (such as Sendmail) consider the two the same, but we mentioned that newer versions of Sendmail do differentiate between the two.

As a simple test we can connect to a newer version of Sendmail and see the difference. (We have removed the extraneous Telnet client output.)

```
sun % telnet vangogh.cs.berkeley.edu 25
220-vangogh.CS.Berkeley.EDU Sendmail 8.1C/6.32 ready at Tue, 3 Aug 1993 14:
59:12 -0700
220 ESMTP spoken here

helo bsdi.tuc.noao.edu
250 vangogh.CS.Berkeley.EDU Hello sun.tuc.noao.edu [140.252.1.29], pleased
to meet you

vrfy nosuchname
550 nosuchname... User unknown

vrfy rstevens
250 Richard Stevens <rstevens@vangogh.CS.Berkeley.EDU>

expn rstevens
250 Richard Stevens <rstevens@noao.edu>
```

First notice that we purposely typed the wrong hostname on the HELO command: bsdi instead of sun. Most SMTP servers take the IP address of the client and perform a DNS pointer query (Section 14.5) and compare the hostnames. This allows the server to log the client connection based on the IP address, not the name that a user might have mistyped. Some servers respond with humorous messages, such as "You are a charlatan," or "why do you call yourself ...". We see in this example that this server just prints our real domain name from the pointer query along with our IP address.

We then type a VRFY command for an invalid name, and the server responds with a 550 error. Next we type a valid name, and the server responds with the username on the local host. Next we try the EXPN command and get a different response. The EXPN command determines that the mail for this user is being forwarded, and prints the forwarding address.

Many sites disable the VRFY and EXPN commands, sometimes for privacy, and sometimes in the belief that it's a security hole. For example, we can try these commands with the SMTP server at the White House:

```
sun % telnet whitehouse.gov 25
220 whitehouse.gov SMTP/smap Ready.

helo sun.tuc.noao.edu
250 (sun.tuc.noao.edu) pleased to meet you.

vrfy clinton
500 Command unrecognized

expn clinton
500 Command unrecognized
```

28.4 SMTP Futures

Changes are taking place with Internet mail. Recall the three pieces that comprise Internet mail: the envelope, headers, and body. New SMTP commands are being added that affect the envelope, non-ASCII characters can be used in the headers, and structure is being added to the body (MIME). In this section we consider the extensions to each of these three pieces in order.

Envelope Changes: Extended SMTP

RFC 1425 [Klensin et al. 1993a] defines the framework for adding extensions to SMTP. The result is called *extended SMTP* (ESMTP). As with other new features that we've described in the text, these changes are being added in a backward compatible manner, so that existing implementations aren't affected.

A client that wishes to use the new features initiates the session with the server by issuing a EHLO command, instead of HELO. A compatible server responds with a 250 reply code. This reply is normally multiline, with each line containing a keyword and an optional argument. These keywords specify the SMTP extensions supported by the server. New extensions will be described in an RFC and will be registered with the IANA. (In a multiline reply all lines except the last have a hyphen after the numeric reply code. The last line has a space after the numeric reply code.)